Manejo de la sesión

Sesión Hijacking

Un atacante logra extraer la cookie de sesión. Se puede obtener aprovechando las siguientes vulnerabilidades:

- Transmión de cookie por texto plano a través de medios inseguros
- Fijación de la sesión
- Inyección de javascript

Esto se puede mitigar tomando las siguientes medidas:

- No transmitir cookies a través de medios inseguros, usar HTTPS
- No permitir el acceso desde javascript a la cookie de sesión con el atributo HTTPOnly
- Hacer que la sesión caduque para evitar que la cookie pueda ser usada indefinidamente.

Session Fixation

Se trata de una vulnerabilidad que consiste en que el atacante consigue que el usuario se identifique con un identificador de sesión que el propio atacante ha generado. Suele seguir los siguientes pasos:

1. El atacante accede a la pagina de inicio de sesión
2. El servidor crea un identificador de sesión y se lo asigna a esa sesión
3. El atacante logra que la víctima abra una URL con el identificador de sesión que ha obtenido previamente

```
http://acme.com/<script>document.cookie=”sessionid=863F3D316”;</script>
http://acme.com/<meta http-equiv=Set-Cookie content=”sessionid=863F3D316”>
```

1. La victima usa dicho enlace para autenticarse, usando el identificador de sesión del atacante.
2. La victima usa sus credenciales y la sesión pasa a ser una autenticada
3. El atacante puede realizar acciones a nombre de la victima a conocer su identificador de sesión.

Para prevenir este tipo de ataques lo que se suele hacer es generar una nueva cookie de sesión cada vez que el usuario se autentica, de esta forma la cookie de sesión del atacante queda anulada tras la nueva autenticación.

Reescritura de URL

Muchos servidores envía en ID de sesión a través de la url de la siguiente forma:

```
http://www.acme.com/account.html;jsessionid=863F3D316
```

Esto se considera una vulnerabilidad, ya que hace que se pueda obtener la sesión por sustracción.

Esto se puede paliar evitando que este identificador de sesión pueda estar en la cabecera, debe ser tratado como si fuera una contraseña. La cabecera HTTP referer contiene información sobre la web en la que se origina una petición, por lo que si el atacante logra redirigir a la víctima a un sitio web que controle puede capturar la URL con la cabecera que contiene la sesión.

Política de mismo origen

(EXAMEN) Define una serie de reglas para restringir como un documento o sus scripts puede interactuar con los recursos localizados en dominios diferentes. Esta es implementada por el navegador web. 2 Webs tienen el mismo dominio si el protocolo, dominio y puerto son el mismo. Las escrituras entre dominios suelen estar permitidas (enlaces, redirecciones y envío de formularios de tipo XMLHttpRequiest y Fetch API). Los navegadores suelen bloquear las escrituras entre dominios por defecto (Peticiones Ajax por ejemplo). También se pueden imponer restricciones sobre cabezeras HTTP.

JSONP o JSON con Padding es una técnica de javascript para realizar llamadas asíncronas a dominios diferentes. Los script con atributo src tienen permitido apuntar a otro dominio dentro de la política de mismo origen. Si desde un script se accede a algo que devuelva JSON se producirá un error de sintaxis. Los servidores con JSONP permiten las peticiones que envíen un parámentro jsonp o callback. Cuando estos parámetros están presentes en la solicitud, la respuesta del servidor devuelve el contenido JSOn dentor de la función especificada como parámetro. De esta forma se consigue devolver un contenido JavaScript en vez de uno JSON.

Intercambio de recursos de orígenes cruzados (CORS)

CORS es un mecanismo que permite al navegador web enviar peticiones AJAX a dominios diferentes. Se distinguen 2 tipos de peticiones:

CORS en Peticiones Simples

Peticiones HEAD, GET o POST. CORS tiene que estar habilitado en el sitio web. El navegador enviará la cabecera Origin en la petición. Se configura en el servidor web y lo implementa el navegador. En la resuesta el servidor envía varias cabeceras con el prefijo Access-Control-*:

- **Access-Control-Allow-Origin** (Obligatoria): Indica si se permiten las peticiones desde el dominio origen.
- **Access-Control-Expose-headers**: El navegador expondrá más cabeceras a través del método getResponseHeader()
- **Access-Control-Allow-Credentials**: Indica si el navegador debe exponer o no las credenciales

CORS en peticiones Complejas

Peticiones que NO sean HEAD, GET o POST, como por ejemplo, PUT o DELETE y aquellas peticiones que no utilizan cabeceras simples. En este caso el navegador va enviar una petición de preflight y recibirá las siguientes cabeceras:

- **Access-Control-request-Method**: Indica el método que utilizará en la petición entre dominios.
- **Access-Control-Request-Headers**: Indica la lista de cabecerás no estándar que se utilizarán en la petición separadas por comas.

En respuesta a la petición preflight el servidor puede responder con las siguientes cabeceras:

- **Access-Contol-Allow-Origin** (Obligatoria): Indica si están permitidas las peticiones desde el dominio origen. Si esta responde como un NO entonces la petición CORS fallará.
- **Access-Control-Allow-Methords**: Lista de los métodos permitidos en la petición CORS separados por comas.
- **Access-Control-Allow-headers**: Lista de cabeceras no estándar permitidas en CORS separadas por comas.

Cross-Site Request Forgery (CSRF o XSRF)

Es una vulnerabilidad en la que el atacante usa el navegador de la víctima para ejecutar peticiones maliciosas en su nombre sobre una web vulnerable. El contenido malicios viene del servidor hacia el navegador, se explota la confianza del navegador en las peticiones autenticadas. Funciona de la siguiente forma:

1. El usuario se autentica creando un identificador de sesión que se instala en su navegador
2. El atacante envía un enlace al usuario que, usando la sesión del usuario, ejecuta una petición en su nombre.

Un requisito para este tipo de ataque es que el usuario debe estar logueado. Otro requisito es que el atacante debe ser capaz de enviar al usuario una petición maliciosa. Esta vulnerabilidad se aprovecha de que las peticiones HTTP sean repetibles ya que los parámetros que se envían son siempre los mismos, no hay componentes aleatorios que varíen cada vez. Para paliar esto se establece un campo aleatorio en los formularios.

Para prevenir estos ataques se debe comprobar que la cabecera HTTP tenga ciertas cabeceras y añadir un token en las peticiones (CSRF Token) que se debe regenerar cada sesión, de forma que no sea repetible y que el atacante no pueda reproducir el formulario de forma sencilla.

- Analizar cabeceras Origin, Refer y Host para determinar la procedencia. El navegador no permite modificar estas cabeceras con javascript.
- El CRFS Token debe tener las siguientes características:
  - Único por sesión
  - Valor aleatorio de tamaño largo
  - Generado por un algoritmo criptográfico.
- En las aplicaciones con estado el token se puede almacenar en la sesión.
- Las peticiones deben incluir este token como parámetro o en alguna cabecera.
- El servidor compara el token de la petición y el de la sesión de forma que si estos no coinciden se descarta la petición
- El token NO debe ir en la URL
- En algunas ocasiones no es viable almacenar el token de sesión, por lo que se envía una cookie doble (Double Submit Cookie)
- Las cookies con el atributo SameSite permiten especificar que no se envíe la cookie si la petición viene de un servidor diferente.

- Otra medida de seguridad podría ser el uso de doble autenticación.